PRODUCTS | SUPPORT | SHOP | CONTACT US | NEWS | EVENTS | SEARCH

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Corel® Paradox® 8 Native Windows® Controls (NWC)

Windows® includes a number of custom controls which allow developers to easily create objects such as list boxes, combo boxes, etc. Corel® Paradox® 8 has encapsulated a few of these custom controls and has made using them even easier.

Five of these custom controls, called Native Windows Controls (NWC) are found in Corel Paradox 8:

1. Combo Box
2. List Box
3. Spin Box
4. Progress Bar
5. Track Bar

Note: Corel Paradox 8 also includes the Corel Web Server control, which is probably a subject for an entire article in itself.

In this newsletter, we'll create a form on which we'll place each of these controls. We will then attach code in the appropriate places to make the controls work together.

Last modified on: 7/17/00

Graphics by COREL

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Native Windows® Controls

Getting Started

First, we need to **create the form**.

- Right-click the Forms icon on the main toolbar or in the Project Viewer.
- Choose New … and click the Blank button.
- Bring up the Object Explorer.
- Change the name of the form (#Form1) to `frmNWC`.
- Set the Title property (on the Appearance page) to `Native Windows Controls`.

Now is a good time to **save your form**.

- Click File, Save.
- Type a meaningful name for your form, such as `NWC.fsl`.

Next, we'll **place each of the controls onto the form**.

On the standard toolbar, immediately to the left of the Selection tool, there is a button pointing up, on top of a button pointing down.

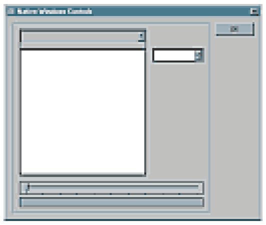
- Click either of these to reveal the Native Windows® Controls (NWC).
- Place a Combo box near the top of your form.
- Next, place a List box beneath the Combo box.
- Now, place a Track bar below the List box.
- Then, place a Progress bar below the Track bar.
- Finally, place a Spin box to the right of the List box.

Let's set some **basic properties**.

Select the Properties tab in the Object Explorer, then set the following properties for each control:

Control	Property	Value
Combo Box	`.Name`	`cboAliases`
List Box	`.Name`	`lbTables`
Trackbar	`.Name`	`tbTrackbar`
Progress Bar	`.Name`	`pbProgress`
Spin	`.Name`	`spnCurrent`

Here's how the form looks so far:



Note that an OK button has been added. It was created by the Button Expert, and simply calls the `::formReturn()` method to close the form. Also, the form is set to Dialog form, and the Minimize and Maximize buttons have been removed.

To do this, set the following properties on the Appearance page:

Property	**Value**
`.DialogForm`	`True`
`.HorizontalScrollBar`	`False`
`.MaximizeButton`	`False`
`.MinimizeButton`	`True`
`.SizeToFit`	`True`
`.ThickFrame`	`True`
`.VerticalScrollBar`	`False`

Back Home Continue

Last modified on: 7/17/00

Graphics by COREL™

PRODUCTS SUPPORT SHOP CONTACT US NEWS EVENTS SEARCH

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Initializing the Form

In the first section, Getting Started, we assigned values to the combo box and the list box, `cboAliases` and `lbTables,` respectively. The `cboAliases` combo box will hold the list of standard aliases currently defined on your system. The `lbTables` list box will list each of the tables contained in the alias selected in the combo box. When you select a table from the list box, the spin box `(spnCurrent)` will contain the index value of the selected table, and the track bar and progress bar will be updated to indicate the current position in the list box. Although this is not the correct use of the progress bar, it will illustrate how to use the progress bar and the other controls.

Now, when we manipulate the spin box, the selected item in the list box will change, as will the positions of the track bar and the progress bar. Manipulating the track bar will affect the list box, the spin box and the progress bar. The progress bar is, of course, “read-only” so we cannot modify it.

Now that we know what we want the form to do, let’s add some code.
Open the code editor to the `frmNWC::Init()` event and add the following code:

```
var
    arAliases array[] string
    stAlias String
    tcAliases Tcursor
    tblAliases Table
endVar

; hide the main Paradox window
appNWC.hide()

; initialize controls
lbTables.reset()
cboAliases.reset()
dynDrivers.empty()
spnCurrent.value = 0
pbProgress.pos = 0
tbTrackBar.pos = 0

; put the alias names into a table and open it
enumAliasNames("AliasInfo")
tcAliases.open("AliasInfo")


; put the aliases into the combo box
while not tcAliases.eot()
    cboAliases.addString(tcAliases.DBName)
    dynDrivers[tcAliases.DBName] = tcAliases.dbType
    tcAliases.nextRecord()
endWhile

; close the TCursor
tcAliases.close()

; delete the table
tblAliases.attach("AliasInfo")
tblAliases.delete()
```

First, we are declaring the number of variables we’ll use to gather the alias names and place them in a table.

The statement `appNWC.hide()` hides the main Paradox window, so that you only see your form on the screen (plus the Windows desktop) when you run the form. `appNWC` is declared as type `Application` in the `::Var()` method of the `frmNWC` form. In the form’s `::Close()` method, a call to `appNWC.show()` will bring the Paradox window back.

Note: the `::Hide()` and `::Show()` methods do not seem to work when switching from Design mode to Run mode. It is strongly recommended that you run the form from the Project Viewer. Otherwise, Paradox disappears into the task list. The only way to recover is to end the task and start again. Be sure to save your form before running it!!

The Native Windows® Controls (NWC) list box and combo box both have a `::reset()` method to clear the object’s contents. The spin box’s main property is the Value property. Setting this to zero (0) effectively initializes the object. The main property of the progress bar and the track bar is the Pos property. Setting the value of this property to zero (0) clears the object’s position.

The combo box uses the `::addString()` method to place values into the control. As we loop through each record in the “AliasInfo” table, we are putting each alias into the combo box, and the table type into the dynamic array `dynDrivers` (declared as type `DynArray[] string` in the Var method of the `frmNWC` form).

Finally, we close the `TCursor` to the table and remove the table, since it is no longer required.

Now, let’s discuss each of the Native Windows Controls.

Back Home Continue

Last modified on: 7/17/00

Tips & Tricks

Native Windows® Controls (NWC)

Corel® Paradox® 8 contains five custom controls, called Native Windows® Controls (NWC):

1. Combo Box
2. List Box
3. Spin Box
4. Progress Bar
5. Track Bar

Back Home Continue

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Last modified on: 7/17/00

Graphics by COREL

COREL
PRODUCTS SUPPORT SHOP CONTACT US NEWS EVENTS SEARCH

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Combo Box

A combo box is similar to a list box in that it contains a list of values. Where it differs is in its presentation on the form. The combo box looks like a simple edit box that contains a button with a down arrow on it. Usually when the combo box gets focus, you can type in a value and the combo box selects the entered value (provided it exists in the list). When you click the down arrow, the list drops down allowing you to scroll through the values and select the one you are interested in. The value selected then appears in the edit box portion of the control. You can only select one value in a combo box.

We now know what a combo box is and how to populate the list (`::addString()` method). What else we can do with the combo box?

The combo box contains a number of unique properties, methods and events.

In the Object Explorer:

- Select the **`cboAliases` control**.
- Select the **Properties tab**.
  Note that there are two properties with a red dot next to the property name. This means that the property is unique to this control.
- Click the **Methods tab**.
  Note that there are five methods with a red dot and note also that each red dot has a padlock. This means that you cannot modify the code behind these methods.
- Click the **Events tab**.
  You'll notice that there is only one event specific to the combo box.

Properties
Methods
Events

Properties

The **`.count` property** tells us how many items are contained in the combo box. To find the number of items in the combo box:

```
var
    lCount LongInt
endVar

    lCount = cboAliases.count
```

Note that the `.count` property returns a `LongInt`. This value cannot be changed.

The **`.itemIndex` property** gives the index of the currently selected item.

```
var
    lSelected LongInt
endVar

    lSelected = cboAliases.itemIndex
```

Again, the `.itemIndex` property returns a `LongInt` value, which is 0-based. Again, you cannot change this value.

Methods

We've already seen the **`::addString()` method** in action. This method simply places the string you specify in the last position in the list. Naturally, the `.count` property is updated to reflect the additional item.

```
cboAliases.addString("Item 1")
cboAliases.addString("Item 2")
cboAliases.addString("Item N")
```

The above statements add three items to the list. If we assume the list was empty (something we should not do), then the `.count` property would return the value 3. “Item 1” is stored in position 0, “Item N” is stored in position 2.

The **`::deleteString()` method** removes an item from the list. It takes a long integer as a parameter. This parameter points to the index of the item to be removed.

```
cboAliases.deleteString(cboAliases.itemIndex)
```

This statement removes the selected item from the list.

The **`::findString()` method** takes a string parameter and returns the index to the item (if it exists).

```
cboAliases.deleteString(cboAliases.findString("Item 2"))
```

If “Item 2” exists in the combo box, the `::findString()` method returns the index value, which is then passed to the `::deleteString()` method.

Use the **`::getText()` method** to get the text of the selected item.

```
msgInfo("Paradox Newsletter", "You’ve selected: " +
    cboAliases.getText(cboAliases.itemIndex))
```

This displays a message box containing the text of the item you’ve selected.

The **`::reset()` method** removes all the list items from the combo box object. It is recommended that you call this method in your form’s `::init()`, since Corel® Paradox® 8 retains the values each time you run your form.

```
cboAliases.reset()
```

Events

The **`::onSelChange()` event** is where you place your code to handle your combo box selections.

```
method OnSelChange()
    msgInfo("Paradox Newsletter", "You’ve selected: " +
        cboAliases.getText(cboAliases.itemIndex))
endMethod
```

Back Home Continue

Last modified on: 7/17/00

Graphics by COREL™

COREL
PRODUCTS SUPPORT SHOP CONTACT US NEWS EVENTS SEARCH

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

List Box

A list box control displays a list of items, and often allows you to select more than one item. The list box supports selecting items by using both the keyboard and the mouse.

Just like the combo box, the list box contains a number of properties, methods and events, some of which are the same as the combo box.

In the Object Explorer:

- Select the **`lbTables` object**.
- Click the **Properties tab**.
  Note that there are four properties unique to the list box.
- Click the **Methods tab**.
  You will see that there are seven methods.
- Finally, click the **Events tab**.
  Three events are revealed.

Properties
Methods
Events

Properties

The **`.count` property** tells you how many items are contained in the list box. You are not allowed to change the value of this property. Corel® Paradox® 8 updates this value as items are added to or removed from the list box.

```
var
    lCount LongInt
endVar

    lCount = lbTables.count
```

The variable `lCount` now contains the number of items contained in the list box.

With the **`.itemIndex` property** you can either determine which item in the list box is selected, or you can tell the list box to select a specific item.

```
var
    lSelected LongInt
endVar

    lSelected = lbTables.itemIndex
    lbTables.itemIndex = lbTables.count - 1
```

In the first line, `lSelected` contains the index of the selected item. The second line selects the last item in the list. Note that the `.itemIndex` property does not return a value if several items are selected. Also, you cannot select several items using this property.

Use the **`.multiSelect` property** and the `.selCount` **property** if you would like to be able to select more than one item from your list box. This property can be set at design time or at runtime.

```
var
    lCount LongInt
endVar

lbTables.multiSelect = true ; turns on multiselect
...
if lbTables.multiSelect then
    lCount = lbTables.selCount
endIf
```

The first statement turns on the multi-select capabilities of the list box. The `if` statement determines if the `.multiSelect` property is set and, if it is, places the number of selected items into the `lCount` variable.

Methods

The **`::addString()` method** simply places the string you specify in the last position on the list. Naturally, the `.count` property is updated to reflect the additional item.

```
lbTables.addString("Item 1")
lbTables.addString("Item 2")
lbTables.addString("Item N")
```

The above statements add three items to the list. If we assume the list was empty (something we shouldn't do), then the `.count` property would return the value 3. "Item 1" is stored in position 0, "Item N" is stored in position 2.

The **`::deleteString()` method** removes an item from the list. It takes a long integer as a parameter. This parameter points to the index of the item to be removed.

```
lbTables.deleteString(lbTables.itemIndex)
```

This statement removes the selected item from the list.

The **`::findString()` method** takes a string parameter and returns the index to the item (if it exists).

```
lbTables.deleteString(lbTables.findString("Item 2"))
```

If "Item 2" exists in the list box, the `::findString()` method returns the index value, which is then passed to the `::deleteString()` method.

Use the **`::getText()` method** to get the text of the selected item.

```
msgInfo("Paradox Newsletter", "You’ve selected: " +
    lbTables.getText(lbTables.itemIndex))
```

This statement displays a message box containing the text of the item you've selected.

The **`::reset()` method** removes all the list items from the list box object. It is recommended that you call this method in your form's `::init()`, since Corel Paradox 8 retains the values each time you run your form.

```
lbTables.reset()
```

This statement removes all entries from the list.

The **`::selRange()` method** allows you to select a range of items (assuming the `.multiSelect` property is set to true) contained in the list. This method takes two parameters: the index of the first item to select and the index of the last item to select.

```
lbTables.selRange(4, 9)
```

This statement selects five items (the 5th through the 10th). Remember that the list is zero-based.

To retrieve the selected items in a multi-select list, use the **`::getMultiSelAsCDL()` method**. Calling this method returns a comma-delimited string which contains the index values of the selected items.

```
var
    sSelItems String
    arSelItems Array[] LongInt
    lLoop LongInt
endVar

sSelItems = lbTables.getMultiSelAsCDL()
sSelItems.breakApart(arSelItems, ",")
for lLoop from 1 to arSelItems.size()
    MsgInfo("Paradox Newsletter", "You’ve selected:" +
        lbTables.getText(arSelItems[lLoop]))
endFor
```

The first statement here puts the indexes of the selected items in to the `sSelItems` variable. The second statement takes the string and places the contents into an array of `LongInt`. The `For` loop simply shows a message box containing the text of each selected item. This is where you would do something useful with the selected items. Typically you would attach this code to a button's `::pushButton()` method.

Events

The **`::onDblClick()` event** occurs when you double-click the list box. In this method you would typically write some code to do something with the currently selected item.

```
msgInfo("Paradox Newsletter", "You’ve selected: " +
    lbTables.getText(lbTables.itemIndex))
```

To get the current selected item, use the `.itemIndex` property.

The **`::onKeyDown()` event** occurs when the list box has focus and a keyboard key is pressed. The virtual key code of the pressed key is sent to this event so that you can handle any specific keys you may be interested in.

```
switch
```

```
    case vkey = VK_RETURN : lbTables.onDblClick()
    case vkey = VK_ESCAPE : cmRebuildList()
endSwitch
```

The first statement checks to see if the Enter (Return) key was pressed. If so, the `::onDblClick()` method is called. This enables the user to choose an item from the list box by pressing the Enter key when an item is selected. The second statement calls a custom method which, presumably, rebuilds the list contents whenever the Escape key is pressed.

`VK_RETURN` and `VK_ESCAPE` are virtual key constants defined by Corel Paradox 8. Check the Corel Paradox 8 online Help topic, “Keyboard constants” for the list of defined constants.

The **`::onSelChange()` event** occurs when you use the arrow keys to change the selection. This event also occurs when you single-click an item with the mouse. You can use this event to display additional information about the selected item. The `::onSelChange()` event occurs after the `::onKeyDown()` event.

```
var
    tblThis Table
    arFields Array[] String
    sTable String
    dbThis Database
endVar

    try
        if dbThis.open(cboAliases.value) then
            sTable = lbTables.getText(lbTables.itemIndex)
            if tblThis.attach(sTable, dbThis) then
                if tblThis.isTable() then
                    tblThis.enumFieldNames(arFields)
                    arFields.view()
                endIf
                dtblThis.unAttach()
            endif
            dbThis.close()
        endIf
    onFail
        errorShow()
    endTry
```

The above code sample opens the database alias contained in the combo box. Then, the table is opened and the field names are enumerated into an array which is then displayed. Finally, the table and the database are closed.

Back Home Continue

Last modified on: 7/17/00

Graphics by COREL™

COREL
PRODUCTS | SUPPORT | SHOP | CONTACT US | NEWS | EVENTS | SEARCH

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Spin Box

The spin box is typically used to increment or decrement counters. It consists of a static text box with Up and Down arrows at the right edge of the text box. Clicking the Up arrow causes the value in the text box to increase. Clicking the Down arrow causes the value to decrease.

In the Object Explorer:

- Select the **`spnCurrent` object**.
- Click the **Properties tab**.
  You will notice that there are two properties.
- Click the **Methods tab**.
  One method is revealed.
- Click the **Events tab**.
  One event is revealed.

Properties
Methods
Events

Properties

The **`.min` property** can be queried to find the minimum value that the spin box will spin up to. This value cannot be changed.

The **`.max` property** can be queried to find the maximum value that the spin box will spin up to. This value cannot be changed.

The **`.value` property** can be used to change the value in the spin box.

```
var
    lMin, lMax LongInt
    lCurrent LongInt
endVar

    lMin = spnCurrent.min

    lMax = spnCurrent.max
    lCurrent = spnCurrent.value
    if lMax > 50 then
        spnCurrent.value = 50
    endIf
```

The first two statements return the minimum and maximum values respectively. The 3rd statement retrieves the current value in the spin box. The statement inside the `if` statement sets the value of the spin box to 50.

Methods

The **`::range()` method** is used to set the minimum and maximum ranges for the spin box.

```
spnCurrent.range(1, 1000)
```

This statement sets the minimum value to 1, and the maximum value to 1000. In order to use the `.value` property, you must use the `::range()` method to set the minimum and maximum ranges.

Events

The **`::onChange()` event** occurs when you click the Up or Down arrow.

```
msgInfo("Paradox Newsletter",
    "Min = " + String(spnCurrent.min) + "\n" +
    "Max = " + String(spnCurrent.max) + "\n" +
    "Value = " + String(spnCurrent.value))
```

This statement simply reveals the values of the `spnCurrent` spin box’s properties each time the value changes.

Back Home Continue

Last modified on: 7/17/00

Graphics by COREL™

COREL
PRODUCTS SUPPORT SHOP CONTACT US NEWS EVENTS SEARCH

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Progress Bar

The progress bar is typically used to indicate the percentage of completion for a lengthy task. For instance, if you need to copy a number of records from one table to another, you could display a progress bar which informs users of the progress of a task.

In Object Explorer:

- Select the **`pbProgress` object**.
- Click the **Properties tab**.
  Note that there is only one property unique to the progress bar.
- Click the **Methods tab**.
  Two unique methods are revealed.
- Click the **Events tab**.
  You will notice that there are no unique events for the progress bar.

Properties
Methods

Properties

The **`.pos` property** is used to set the progress bar indicator to a specific position within the defined range. You may also query the `.pos` property to retrieve the current position.

```
var
    lPos LongInt
endVar

    lPos = pbProgress.pos
    if lPos = 50 then
        pbProgress.pos = 100
    endIf
```

In this example, the first statement retrieves the current position of the progress bar. Then, if the current position is equal to 50, set the position to 100.

Methods

Use the **`.setRangeAndStep()` method** to initialize the progress bar. This method takes three parameters: the minimum value, the maximum value and the value to increment each time the `.stepIt()` method is called.

The **`.stepIt()` method** is used to increment the progress bar to the next step. Typically, the `.stepIt()` method is called within a loop.

```
var
    lCount LongInt
    lLoop LongInt
    arItems Array[] String
endVar

    lCount = arItems.size()
    pbProgress.setRangeAndStep(1, lCount, 1)
    pbProgress.visible = True

    For lLoop from 1 to lCount
        ; do something with arItems[lLoop]
        pbProgress.stepIt()
    endFor
    pbProgress.visible = False
```

The first statement retrieves the number of items in the array. The second statement sets the minimum position to 1, sets the maximum value to the number of array items, and sets the increment to 1. The progress bar's `.visible` property is set so that it will appear on the form. Inside the `For` loop, something is done with array item, then the progress bar is updated. Finally, the progress bar is hidden again. Typically you would display the progress bar only when you need it.

Back Home Continue

Last modified on: 7/17/00

COREL

PRODUCTS | SUPPORT | SHOP | CONTACT US | NEWS | EVENTS | SEARCH

Tips & Tricks

COREL Paradox 8

- Product Support
- Product Highlights
- System Requirements
- Pricing & Languages
- Learning
- Tips & Tricks
- Developer Support
- Reviews
- Case Studies

Track Bar

The track bar (sometimes called a slider control) is often used to select a value within a defined range.

In the Object Explorer:

- Select the **`tbTrackBar` object**.
- Click the **Properties tab**.
  Note that there are 13 properties unique to the track bar.
- Click the **Methods tab**.
  Note that no methods appear.
- Click the **Events tab**.
  Here you'll see nine events.

Properties
Events

Properties

The **`.EnableSelRange` property** is used to determine whether the track bar has a selectable range. If this property is set to True, then the **`.SelStart`** and **`.SelEnd` properties** can be used to set the selected range.

The **`.LineSize` property** specifies the number of ticks the thumb moves on `::LineUp()` and `::LineDown()` events. The default value is 1.

The **`.Max` property** specifies the maximum value for the track bar range and the **`.Min` property** specifies the minimum value for the track bar range.

```
tbTrackBar.max = arTables.size()
tbTrackBar.min = 1
```

The first statement sets the maximum value of the track bar to the number of items contained in the array `arTables`.

The **`.orientation` property** determines if the track bar will be horizontal (0-default) or vertical (2-vertical). While this property can be set at run-time, it would be best to set it at design time. If you would like to have a vertical track bar, you must draw it vertically on your form.

The **`.PageSize` property** specifies the number of ticks the thumb moves on `::PageUp()` and `::PageDown()` events. The default value is 1.

The **`.Pos` property** indicates the current thumb position. This can be set at run-time. Any event which moves the thumb will update this property.

```
var
    lPos LongInt
endVar

    lPos = tbTrackBar.pos
    tbTrackbar.pos = 1
```

The first statement retrieves the current position of the thumb. The second statement moves the thumb to the first position.

The **`.SelEnd` property** indicates the end of the selected range, and the **`.SelStart` property** indicates the start of the selected range. These properties are meaningful only when the `.EnableSelRange` property is set to True.

The **`.Style` property** allows you to set several properties which describe the appearance of the track bar.

The **`.TickFrequency` property** specifies the spacing between each tick mark. For example, if you set this property to 2, a tick mark will appear in every second position.

```
tbTrackBar.TickFrequency = 2
```

The **`.tickMarks` property** determines where the tick marks will appear. When the value is set to 0 (default), the tick marks are on the bottom (horizontal) or on the right (vertical) of the thumb. When the value is set to 4, the tick marks are on the top (horizontal) or on the left (vertical) side of the thumb. When the value is set to 8, the tick marks are both on the top and bottom or on both the left and right side of the thumb.

With the **`.tickStyle` property**, you can specify whether to display the tick marks. Setting this property to 0 tells the track bar not to display tick marks. Setting the property to 1 will display tick marks.

Events

The **`::EndTrack()`** `event` is the last event to occur when the track bar has been manipulated. This event can be considered the equivalent of a generic `::onChange()` event.

The **`::LineDown()`** `event` is generated when the down key is pressed (when the track bar has focus). The track bar's thumb position decrements by the value in the `.LineSize` property.

The **`::LineUp()`** `event` is generated when the up key is pressed. The thumb position increments by the value of the `.LineSize` property.

The **`::MoveBottom()`** `event` is generated when the End key is pressed. The thumb position moves to the track bar's maximum value.

The **`::MoveTop()`** `event` is generated when the Home key is pressed. The thumb position moves to the track bar's minimum value.

The **`::PageDown()`** `event` is generated when either the PageDown key is pressed, or when the track bar is clicked to the right of or below the thumb (depending on horizontal or vertical position). The thumb position is incremented by the value in the `.PageSize` property.

The **`::PageUp()`** `event` is generated when either the PageUp key is pressed, or when the track bar is clicked to the left of (horizontal orientation) or above the thumb (vertical orientation). The thumb position is decremented by the value in the `.PageSize` property.

The **`::ThumbPosition()`** `event` occurs as soon as the left mouse button has been released after the thumb has been dragged with the mouse.

The **`::ThumbTrack()`** `event` occurs while the thumb is being dragged with the mouse. Releasing the mouse button generated a `::ThumbPosition()` event.

Back Home Continue

Last modified on: 7/17/00

PRODUCTS SUPPORT SHOP CONTACT US NEWS EVENTS SEARCH

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Putting It All Together

Now that we know all the properties, methods and events for the Native Windows® Controls, let's finish our form.

We've already created the `frmNWC::Init()` event and there has been reference to the variables which should be declared in the `::Var()` method. If you haven't put in this code, do it now.

```
Var
    dynDrivers DynArray[] String
    appNWC Application
endVar
```

The `DynArray` field will contain the alias type for each of the aliases you've loaded into the combo box; you will use this later. Use the `Application` variable to hide (and restore) the Corel® Paradox® 8 window.

Now, the form's `::Close()` event.

```
method close(var eventInfo Event)

    if eventInfo.isPreFilter() then
        ; This code executes for each object on the form
    else
        ; This code executes only for the form
        appNWC.show()
    endIf

endMethod
```

When you close the form, you want Corel Paradox 8 to reappear. If you don't complete this step, the only way to shut down Corel Paradox 8 is to end the task in Task Manager.

The form is now complete. Next we'll handle the combo box event:
the `cboAliases::OnSelChange()` event.

```
method OnSelChange()

var
    arTables Array[] String
    stTableName String
    lLoop LongInt
    stDriver String
endVar

    lbTables.reset()
    stDriver = dynDrivers[cboAliases.value]

    if stDriver = "STANDARD" then
      enumDatabaseTables(arTables, cboAliases.value, "*.db")
    else
      enumDatabaseTables(arTables, cboAliases.value, "")
    endif

    for lLoop from 1 to arTables.size()
      lbTables.addString(arTables[lLoop])
    endFor

    spnCurrent.range(1, arTables.size())
    spnCurrent.value = 1
    pbProgress.setRangeAndStep(1, arTables.size(), 1)
    pbProgress.pos = 1
    tbTrackBar.min = 1
    tbTrackBar.max = arTables.size()
    tbTrackBar.pos = 1

endMethod
```

Start by removing all entries from the `lbTables` list box. Then, obtain the database driver and, if you've selected a Corel Paradox 8 database, look for .db files. Otherwise, look for other table types. In the `For` loop, place each table into the list box. Finally, set the minimum and maximum properties for the spin box, the progress bar and the track bar.

Next, the list box. Here is the code for the `lbTables::OnSelChange()` event:

```
method OnSelChange()

    spnCurrent.value = self.ItemIndex + 1
    pbProgress.pos = self.ItemIndex + 1
    tbTrackBar.pos = self.ItemIndex + 1

endMethod
```

In this event, assign the index of the selected item to the position properties of the spin box, the progress bar and the track bar. We need to increment these values because the list box is zero-based, and the spin box, progress bar and track bar are one-based.

The event handler for the `spnCurrent::OnChange()` event is almost as simple:

```
method OnChange()

    lbTables.ItemIndex = self.value - 1
    pbProgress.pos = self.value
    tbTrackBar.pos = self.value

endMethod
```

Take the value from the spin box, and place it in the `.ItemIndex` property of the list box, which changes the selected item. Then, place the spin box value and use it to update the progress bar and the track bar.

The progress bar does not have any events so you don’t have to respond to anything. The track bar, on the other hand, has many events; however, you only need to respond to one. Here is the `tbTrackBar::EndTrack()` event:

```
method EndTrack()

    lbTables.ItemIndex = self.pos - 1
    spnCurrent.value = self.pos
    pbProgress.pos = self.pos

endMethod
```

These statements take the position of the track bar and change the selected item in the list box, as well as update the spin box and the progress bar.

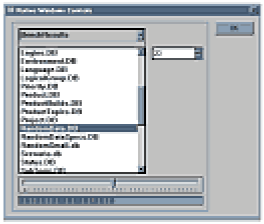
Back Home Continue

Tips & Tricks

COREL Paradox 8
Product Support
Product Highlights
System Requirements
Pricing & Languages
Learning
Tips & Tricks
Developer Support
Reviews
Case Studies

Running the Form

Now that we've finished coding the form, we need to save it and run it. Here's what the form looks like:



As you select items in the tables list box, you will notice the spin box, the track bar and the progress bar update to reflect the currently-selected table. Play with this form and do some experimenting on your own to gain an understanding of how the Native Windows® Controls (NWC) can work for you in your projects.

Back Home

Last modified on: 7/17/00

Graphics by COREL